# `MultiJoin`: An R package for efficiently joining files on disk

*by Markus Loecher*

**Abstract** Database joins of multiple tables on common keys are a powerful paradigm. In the absence of a database or for very large tables we propose an efficient disk based merge based on the Unix *join* utility. A combination of file compression, named pipes and regular Unix pipes allows the user simulateneous join queries on multiple tables. The `R` package `MultiJoin` provides easy-to-use wrapper functions which launch such compound join queries.

## Review of Unix join

The Unix *join* utility scans two files at once and for each pair of input lines with identical join fields, writes a merged line to standard output. The join field can be specified and one of the files can b replaced by standard input. Note that both files must be sorted on the join fields.

We distinguish between *left joins* which print only lines with common keys ("intersection") and *full joins* where every unique key value produces a line ("union"). The "-a" option activates the full join by also printing unpairable lines We present two use cases:

1. Multivariate spatiotemporal data such as weather recordings. Assume one file per hour containing $M$ measured variables such as temperature, air pressure, precipitation, etc. (columns) for $N$ locations (rows). We assume the latter to be the key.
2. Word count files from Google Ngram
3. User demographics

## The FullJoin function

The main functionality of the package is provided by the flexible *FullJoin* function which can handle compressed files as well as desired manipulations during the join such as selecting columns or rows, sorting, applying regular expressions, etc. For the following examples, we adopt simple file names such as ftr1.txt, ftr2.txt... If the files on disk do not have to be uncompressed or filtered in any way the following call performs a full join on 4 files each of which contains 3 columns, the first of which is the key.

```
FullJoin(files=paste0("ftr",1:4,".txt"), NumFields = rep(3, 4),missingValue="0", suffix = "")
```

producing

```
  join  -a 1 -a 2 -o "0 1.2 1.3 2.2 2.3" -e 0 -1 1 -2 1 ftr1.txt ftr2.txt
| join  -a 1 -a 2 -o "0 1.2 1.3 1.4 1.5 2.2 2.3" -e 0 -1 1 -2 1 - ftr3.txt
| join  -a 1 -a 2 -o "0 1.2 1.3 1.4 1.5 1.6 1.7 2.2 2.3" -e 0 -1 1 -2 1 - ftr4.txt
```

The next example illustrates how named pipes (FIFOs) can be used to uncompress and filter files before sending them to the join command chain. Here, we only keep columns 1 (the key) and 3:

```
FullJoin(paste0("ftr",1:3,".txt.gz"), mycat = "gunzip -cf ", filterStr = " | cut -f1,3", suffix = "")
```

producing

```
  gunzip -cf  ./ftr1.txt.gz  | cut -f1,3  >  /tmp/fifo1  &
  gunzip -cf  ./ftr2.txt.gz  | cut -f1,3  >  /tmp/fifo2  &
  gunzip -cf  ./ftr3.txt.gz  | cut -f1,3  >  /tmp/fifo3  &
  join  -a 1 -a 2 -o "0 1.2 2.2" -e NA -1 1 -2 1 /tmp/fifo1 /tmp/fifo2
| join  -a 1 -a 2 -o "0 1.2 1.3 2.2" -e NA -1 1 -2 1 - /tmp/fifo3
```

Note the default use of the `-e NA` option which specifies NA to be the filling character for missing values. Specifying the argument `FullJoin(...,missingValue="0")` changed that to a zero in the previous example. We can also add a suffix which would for example compress the result from the compound join and write to a file.

```
FullJoin(paste0("ftr",1:3,".txt.gz"), mycat = "gunzip -cf ", filterStr = " | cut -f1,3", suffix = " | gzip >
```

Alternatively, we can directly read the output into a data structure within R.

```
x=  FullJoin(paste0("ftr",1:3,".txt.gz"), mycat = "gunzip -cf ", filterStr = " | cut -f1,3", suffix = " ")
```

## Benchmarks

We can conveniently benchmark the function execution by adding the `prefix = "time"` option to the call or -alternatively- use the R function `proc.time()`. Figure 1 shows the results for joining a varying

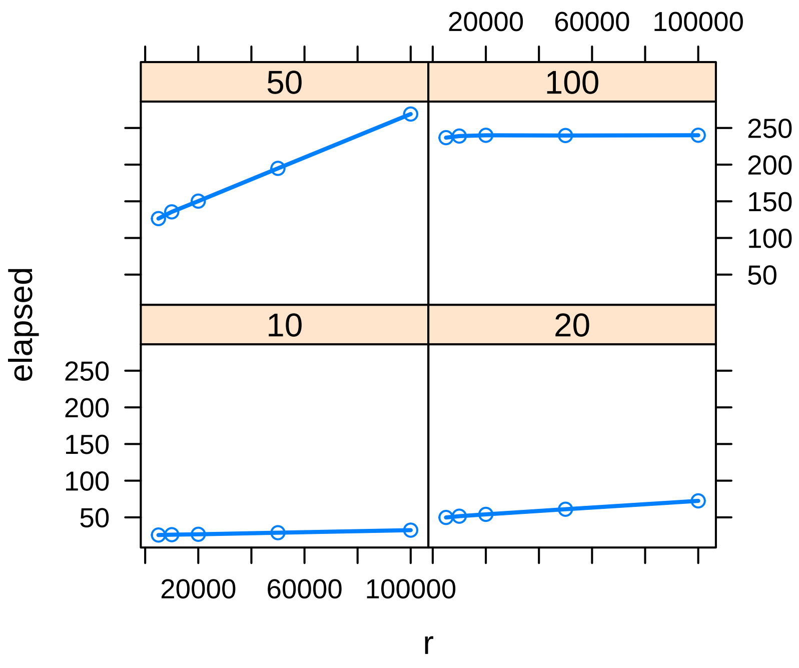

**Figure 1:** CPU time taken to join 10/20/50/100 compressed files, respectively, as a function of the number of rows. The number of columns are 6 for each file and the elapsed time is measured in seconds.

number of compressed files and writing the output to a temporary file. Both slope and intercept seem to depend on the number of files.

## Summary

For situations where data are accesssed infrequently and hence typically stored on disk, the R package `MultiJoin` provides easy-to-use wrapper functions which launch compound join queries. We utilize the efficiency of the Unix join command and the convenience of Unix pipes and FIFOs to chain the intermediate outputs together. We hope to fill a niche where users both (i) abstain from unsing a full database solution and (ii) it is not feasible to read the data into R and then join with e.g. the R package `data.table`.

*Markus Loecher*
*Berlin School of Economics and Law*
*Badensche Str. 52, 10825 Berlin*
*Germany*
mloecher@hwr-berlin.de